Project 3: *Autoencoder dimensionality reduction, EMD-Manhattan metrics comparison and classifier based clustering on MNIST dataset*

Team Members:

1. Dimitrios Konstantinidis (sdi1700065@di.uoa.gr)
2. Andreas Spanopoulos (sdi1700146@di.uoa.gr)

A little bit about our work

In this project we implement 4 different evaluations regarding dimensionality reduction from an autoencoder model, comparison in NN search between original item space, reduced item space and LSH application on the original space, comparison between the Wasserstein and Manhattan metrics on NN, comparison between centroid-based clustering through k-medians with classifier based clustering.

1. The program reduce.py accepts as input a dataset and using a pretrained encoder produces a new dataset based on the latent vector output of the encoder.
2. The program search.cpp accepts the original dataset input and the modified input from reduce.py and executes a comparison between Brute Force NN search in the original space, Brute Force NN search in the reduced space and LSH ANN search in the original space.
3. The programs manhattan.cpp and emd.py accept as inputs a dataset, a queryset and their respective labels, then they each evaluate their own metric based on the amount of same-label neighbors returned by a kNN search.
4. The program cluster.cpp accepts the original dataset input, the modified input from reduce.py and the predictions made by classification.py to produce 3 different cluster models, afterwards it calculates some valuable metrics on each to evaluate their performance.

Part 1 - Dimensionality Reduction

As stated before, this program converts a given dataset to a reduced version of it by passing it through a pretrained encoder model and taking the latent vector values produced by it. Analytically, the first part of this program is to create a viable encoder model, helpfully, the autoencoder directory contains a program which trains an autoencoder on a given dataset and allows the user to save the encoder part of the model. Hence, we would strongly advise the encoder used to be produced from this specific program (Note: the notebook directory contains an almost identical colab notebook implementation of the autoencoder for easier GPU usage). After the encoder model has been saved, we are free to reduce the dimensionality of our dataset. The reduce.py file contains a local variable pointing to the encoder to-be-used which can be changed to specify the user's preference. Essentially, the dataset and queryset given as arguments are passed through the encoder model, flattened and then written to the respective output files (given as arguments).

Part 2 - Original Space & Latent Space brute force NN and LSH ANN on original space comparison

This program accepts as input a dataset and a queryset both in the original and the reduced dimension versions of them (i.e. the output of **Part 1**). Then makes comparisons between: - Brute Force NN on original space. - Brute Force NN on reduced space. - LSH ANN on original space.

With respect to the brute force NN on original space, approximation factors of the latter 2 approaches are calculated (based on relative distance of neighbor produced). Essentially, the original dataset is loaded into both a BruteForce class and an LSH class. Alongside them, the reduced dataset is loaded into a BruteForce class. Afterwards, for each query image of the respective queryset (original/reduced) the Nearest Neighbor is calculated on all 3 models. Finally, all these calculations are written to an output file where some extra information is recorded (approximation factors, times etc.).

Part 3 - Earth Mover's Distance & Manhattan Distance metrics comparison

In this part, we are evaluating the Earth Mover's Distance (Wasserstein Distance) against the Manhattan Distance as metrics. This is split into two programs. The first program lies inside the NN_Clustering directory and is an evaluation on the Manhattan Distance. The second program lies on the EMD directory and is a python script regarding the evaluation of the EMD metric. The reason we have two programs is that the Manhattan distance (which is supposed to be fast) was taking way too long on python whereas in C++ with the O3 flag during compilation, the results timewise were great. On the EMD part, we created a python script instead of modifying the aforementioned Manhattan C++ Program as the tools for linear programming in C++ are a pain to use properly.

The evaluation was done by taking the sum of all same-labeled neighbors returned by kNN (on each metric) divided by all neighbors returned by kNN.

The Manhattan C++ Program is pretty simple in its contents as it uses code from the BruteForce class used in **Part 2**. Hence, the implementation can be easily distinguished from inside the file.

Regarding the implementation of the EMD metric, the approach is pretty straightforward. We start by dividing the given image into clusters, that is, we partition the image into sub-images. We calculate the signature of every cluster (sub-image), as the sum of its normalized pixel values. We also calculate the Euclidean distance of the clusters. Having these features calculated, we have prepared the foreground for computing the EMD metric.

The EMD metric is the (normalized) value of the optimal solution of the Transportation Problem that is generated by considering the cluster signatures of one image as the supply, and the cluster signature of the second image as the demand. The costs of the objective function are basically are the Euclidean Distances mentioned above. There are many clever algorithms to solve this minimization problem (e.g. U-V Method, aka Reduced Costs method), which is basically the Simplex algorithm slightly modified to reduce operations and therefore Complexity. For simplicity, we compute the optimal solution by using the OR-Tools Python Library.

Part 4 - Centroid-based clustering on original & latent space and classifier based clustering comparison

The program that implements this comparison is cluster.cpp. It accepts as arguments the dataset in the original space, in latent space, a configuration file (containing parameters regarding the clustering like number_of_clusters etc.), a path to the output file and a file containing grouped images per label from the dataset produced

by a classifier. We have implemented a classifier like such that based on a pre-trained encoder classifies the images of the dataset based on the digits they represent. This file's format is:

```
CLUSTER-label {size: amount_of_image_ids, image_id_1, image_id_2, ..., image_id_n}
CLUSTER-next_label {....}
```

You are free to use your own file (as long as it complies with the specified format). To use our classifier, at first you need to have a pre-trained encoder saved. The one created by the autoencoder used in **Part 1** works perfectly.

The clustering executable, produced from the cluster.cpp file, uses the Clustering class which is defined in the clustering.hpp header. The Clustering algorithm used is k-medians. We use the Clustering procedure on both the original dataset and its reduced version. Then, the Clustering of the reduced space gets translated in a Clustering for the original space. It is not as accurate as the direct Clustering in the original space, but it achieved much faster convergence. The silhouette values computed in the end, also suggest the above conclusion. The Clustering algorithms is implemented as follows: - If no pre-assigned clusters are provided for each data point, then the centroids get initialized using the k-means++ algorithm. Then, Lloyd's algorithm is used for the Assignment Step. The update step is that of the k-medians algorithm. - If pre-assigned clusters are provided for each data point, then the data points get placed in their corresponding clusters, and then the new coordinates (components) of the cluster get computed normally, as if we had a regular update step in the k-medians algorithm.

After the Clustering algorithms have been executed, their results gets logged in the specified output file, along with some other information regarding the whole procedure.

Usage

Part 1

To execute the autoencoder.py program (to produce the encoder), type:

```
$ python3 autencoder.py -d dataset
```

You will then be prompted for the location of the encoder model to be saved at. There is also the option to use the colab notebook in the notebook directory.

To specify the encoder in the reduce.py program, change the local variable named "encoder_path" with the path of the encoder you want to use. To execute:

```
$ python3 reduce.py -d dataset -q queryset -od output_dataset_file -oq output_query_file
```

Then, the dataset & queryset produced files will be created to be used for **Part 2**

Part 2

To execute the search program, first make sure to use

```
$ make SEARCH
```

inside the NN_Clustering directory. Afterwards, use the following command to run the program

```
$ bin/search  –d input_file_original_space
              -i input_file_new_space
              –q query_file_original_space
              -s query_file_new_space
              –k number_of_LSH_hash_functions
              -L number_of_LSH_hash_tables
              -o output_file
```

Then, the file containing information about the experiment will be produced to the path specified.

Part 3

To compare the two evaluations, you need to at first run the manhattan program. Navigate to the NN_Clustering directory and compile it by typing:

```
$ make MANHATTAN
```

Afterwards, you can execute it by typing:

```
$ bin/manhattan   –d input_dataset
                  –q query_dataset
                  –l1 input_dataset_labels
                  -l2 query_dataset_labels
                  -o output_file
```

Once finished, the average correct search results will be both printed on the terminal and written in the provided output file path.

For the EMD evaluation program as it is a python script, there is no need for compilation, we simply have to navigate to the EMD directory and type:

```
$ python3 emd.py  –d input_dataset
                  –q query_dataset
                  –l1 input_dataset_labels
                  -l2 query_dataset_labels
                  -o output_file
```

Once finished, the average correct search results will be both printed on the terminal and written in the provided output file path. **Notice**: The results of EMD will be appended on the specified output file so if you want to store both results in a single output file, make sure you include the same output file in the arguments of both program (also, you have to run the manhattan program first as it truncates the output file provided at start).

Part 4

To execute the classifier so as to produce the cluster file, type:

```
$ python3 classifier.py -d dataset -dl dataset_labels -ol output_file -model encoder
```

inside the classifier directory. Keep in mind that there is also the option to use the colab notebook in the notebook directory which contains practically the same code but it makes the usage of a GPU much easier. Afterwards, the cluster file will have been produced in the output_path specified (or through the notebook to the place provided by the user there) so we are free to execute the clustering program. First, make sure to use

```
$ make CLUSTER
```

inside the NN_Clustering directory. Afterwards, use the following command to run the program

```
$ bin/cluster –d input_file_original_space
              -i input_file_new_space
              –n cluster_file
              -c configuration_file
              -o output_file
```

Then, the file containing information about the experiment will be produced to the path specified.

Analysis

Under the analysis directory, there exist some experiments we have ran for all four parts of this project. Keep in mind they are not extensive and in some parts, some degree of randomness exists (in LSH for example) so you may end up with different results.

GitHub Repository